**Setting up a server for data and stats**

I chose Amazon Web Services Elastic Cloud to host the server. I used the Elastic IP service (fairly cheap) to allocate a permanent IP address to the instance.

I was keen on using Red Hat or similar flavoured Linux, because of purpose #4 noted above; I already knew a bit about Ubuntu and wanted to expand my knowledge sphere, and Red Hat is the flavour that seems to pop up on corporate and government servers. In the end I opted for CentOS, which is [“a Linux distribution that provides a free, enterprise-class, community-supported computing platform functionally compatible with its upstream source, Red Hat Enterprise Linux (RHEL)”](https://en.wikipedia.org/wiki/CentOS). In fact, installing R is a bit easier on CentOS that it was on my Red Hat experiments.

I want the following on this machine:

* R, RStudio Server and Python 3 for analysis and for data munging
* PostgreSQL for storing data and supporting analysis
* Shiny Server for dissemination
* A webserver (I chose Nginx) to support RStudio Server and Shiny Server through reverse proxy so I can access them on regular web browser ports rather than ports 8787 and 3838, which will often not be available from a corporate network
* all the other utilities and extras to support all this, such as curl, mail and fonts

This was non-trivial, which is one of the reasons why I’m blogging about it! There are lots of fishhooks and little things to sort through and while there are some excellent blog posts and tutorials out there to help do it, none of them covered quite the end-to-end setup I needed. One of my outputs from the process was a set of notes – not quite a single run-and-forget configuration script as you’d want in a professional setup, but fairly easy to use – that makes it easy to do similar in the future.

**R and a few basics**

Here’s where I started, once having started up the server (plenty of tutorials on how to do that provided by Amazon themselves and others). I start by installing R, including the epel “Extra Packages for Enterprise Linux” that are needed beforehand.

# we'll want mail later

sudo yum install -y mailx

#-----------------R and its dependencies------------------------

#install R. We need epel first

sudo yum update -y

sudo yum install –y https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm

sudo yum install -y texlive

sudo yum install -y texinfo

# clean up

rm \*.rpm

sudo yum install -y R

# and version control of course

sudo yum install -y git

**Database**

Next thing was to install PostgreSQL. I found it useful to install this before I started installing R packages, because some R packages that speak to PostgreSQL behave differently on installation depending on whether PostgreSQL is found on the machine or not

#=====================postgresql=======================

# install postgresql. Good to do this before we start installing R packages

sudo yum install -y postgresql-server postgresql-contrib postgresql-devel

sudo postgresql-setup initdb

# stop to give postgres account a password for the operating system

sudo passwd postgres

# start the postgresql service

sudo systemctl start postgresql

# I think this next line means the database service restarts when the machine is rebooted

sudo systemctl enable postgresql

**Extending R via packages and their dependencies**

I find installing all the R packages I regularly use a harder job in Linux than Windows. I’m sorry, but I do. In particular, Windows installations of packages like gdal seems to look after upstream dependencies seamlessly and quietly. Not so on Linux. Here’s what I needed to do at the command line to get all the R packages I wanted installed.

#======================miscellaneous dependencies needed by R packages===============

# iBest to do this on a large instance, even if you only start it as big

# during the install. 8GB seems a minimum

#----------------tidyverse and Cairo---------

# First, some dependencies that rvest, devtools, Cairo need:

sudo yum install -y libcurl-devel libxml2-devel openssl-devel

sudo yum install -y cairo-devel libXt-devel udunits2-devel gdal-devel poppler-cpp-devel

#------------The gdal epic---------------------

# needed for spatial stuff and in particular sf which needs version > 2 (currently 2.2).

# This should work according to the sf github page (excluding udunits2 which we look after later):

sudo yum install -y gdal-devel proj-devel proj-epsg proj-nad geos-devel

# but that installs the wrong version of gdal! We have to install it by hand.

# see https://gis.stackexchange.com/questions/263495/how-to-install-gdal-on-centos-7-4

# adapted from https://gist.github.com/simondobner/f859b2db15ad65090c3c316d3c224f45

wget http://download.osgeo.org/gdal/2.2.4/gdal-2.2.4.tar.gz

tar zxvf gdal-2.2.4.tar.gz

cd gdal-2.2.4/

./configure --prefix=/usr/ --with-sfcgal=no

make -j4

sudo make install

# should have a test here to only do the next two things if installed correctly!

rm \*.tar.gz

rm gdal-2.2.4 -r

#======================R packages=====================

# Now we can install R packages that need all the above system dependencies first.

#

# udunits2 needs special configuration when installing in R so let's do that first and get it out of the way

sudo R -e "install.packages('udunits2',configure.args='--with-udunits2-include=/usr/include/udunits2', repos='http://cran.rstudio.com/')"

# these are a bunch of packages that are heavily used and that I want installed up front and available

# for all users (hence installing them as super user)

sudo R -e "install.packages(c('Rcpp', 'rlang', 'bindrcpp', 'dplyr', 'digest', 'htmltools', 'tidyverse',

'shiny', 'leaflet', 'sf', 'scales', 'Cairo', 'forecast', 'forcats', 'h2o', 'seasonal', 'data.table',

'extrafont','survey', 'forecastHybrid', 'ggseas', 'treemap', 'glmnet', 'ranger', 'RPostgres', 'igraph',

'ggraph', 'nzelect', 'tm', 'wordcloud', 'praise', 'showtext', 'ngram', 'pdftools', 'rtweet', 'GGally',

'ggExtra', 'lettercase', 'xgboost'), repos='http://cran.rstudio.com/')"

# fonts

sudo yum install dejavu-sans-fonts

sudo yum install -y google-droid-\*-fonts

sudo yum install -y gnu-free-\*-fonts

sudo R -e "extrafont::font\_import(prompt = FALSE)"

I did all of this with my server set up as a “large” instance with 8GB of RAM. This particularly makes a difference when installing Rcpp. After all the initial is setup you can stop the instance, downsize it something cheaper, and restart it.

Note that I am using sudo to install R packages so they are available to all users (which will include the shiny user down the track), not just to me. I wanted everyone using this server to have the same set of packages available; obviously whether this is desirable or not depends on the purpose of the setup.

**Server-related stuff**

Next I want to get RStudio Server and Shiny Server working, and accessible via a web browser that just talks to standard port 80. There is a step here where the Nginx configuration file gets edited by hand; the links to RStudio support for RStudio Server and for Shiny Server contain instructions on what needs to go where.

Also note that the actual versions of RStudio Server and of Shiny Server below are date-specific (because they are installed via local install), and probably the links are already out of date.

#-------------webby stuff------------

# install a web server so we can deliver things through it via reverse proxy

# see https://support.rstudio.com/hc/en-us/articles/200552326-Running-RStudio-Server-with-a-Proxy

# and https://support.rstudio.com/hc/en-us/articles/213733868-Running-Shiny-Server-with-a-Proxy

sudo yum install -y nginx

#install RStudio-Server (2018-04-23)

wget https://download2.rstudio.org/rstudio-server-rhel-1.1.447-x86\_64.rpm

sudo yum localinstall -y --nogpgcheck rstudio-server-rhel-1.1.447-x86\_64.rpm

#install shiny and shiny-server (2018-04-23)

wget https://download3.rstudio.org/centos6.3/x86\_64/shiny-server-1.5.7.907-rh6-x86\_64.rpm

sudo yum localinstall -y --nogpgcheck shiny-server-1.5.7.907-rh6-x86\_64.rpm

rm \*.rpm

# now go make the necessary edits to /etc/nginx/nginx.conf

# note that the additions are made in two different bits of that file, you don't just past the whole

# lot in.

sudo nano /etc/nginx/nginx.conf

sudo systemctl restart nginx

# go to yr.ip.number/shiny/ and yr.ip.number/rstudio/ to check all working

# add some more users if wanted at this point

# sudo useradd ellisp

# sudo passwd ellisp

# not sure if all these are needed:

sudo systemctl enable nginx

sudo systemctl enable rstudio-server

sudo systemctl enable shiny-server

# set the ownership of the directory we're going to keep apps in so the `shiny`

# user can access it

sudo chown -R shiny:shiny /srv/shiny-server

**Python**

Centos currently comes with Python 2.7, but I wanted to be using Python 3. My Python skills are halting at best but I want them to be as future-proofed as possible. Anaconda seems a relatively straightforward way to manage Python.

#---------------Anaconda / python----------------

# go to https://repo.continuum.io/archive/ or https://www.anaconda.com/download/#linux to see the latest version

# Anaconda3 is with python 3.X, Anaconda2 is wit python 2.7. Note

# that python 2.7 is part of the Centos linux dsitribution and shouldn't be

# overwritten ie python xxx.py should run python 2.7. But doing the process below does this;

# watch out for if this causes problems later...

#

wget https://repo.continuum.io/archive/Anaconda3-5.1.0-Linux-x86\_64.sh

sudo bash Anaconda3-5.1.0-Linux-x86\_64.sh

# agree to the license, and specify /opt/anaconda3 as location when asked

# we want to give all users anaconda on their path, so I snitched this from:

# https://www.vultr.com/docs/how-to-install-jupyter-notebook-on-a-vultr-centos-7-server-instance

sudo cp /etc/profile /etc/profile\_backup

echo 'export PATH=/opt/anaconda3/bin:$PATH' | sudo tee -a /etc/profile

source /etc/profile

echo $PATH

sudo /opt/anaconda3/bin/conda conda install psycopg2

# as far as I can tell this makes python3.6 the default python, which is surely going to cause problems down

# the track...

**Configuring PostgreSQL**

I installed PostgreSQL and started its database service early in this process, but in the next step need to actually set up some database and users for use. The PostgreSQL security model is thorough and comprehensive but with lots of fishhooks. Here’s how I set it up for this particular (very simple) use case. First, I enter the psql environment as the postgres user (currently the only user with any access to the database server)

sudo -u postgres psql

Now we can set up the users we want to be accessing our databases; some databases for them to use; and schemas within those database. In this case, I set up two databases for now

* survey\_microdata
* twitter

and three different users, in addition to postgres:

* ellisp (ie me, in development mode)
* external\_analyst (ie me or others, in read-only mode)
* shiny (the Shiny Server’s id on the server, needed so Shiny apps can access the database)

-- you are now in psql as user postgres. Although default is to use unix's identification of you,

-- and you don't need a password to access the database from the local host, it's good to have a

-- password if you want to set up other connections later

\password postgres

CREATE DATABASE survey\_microdata;

CREATE DATABASE twitter;

CREATE ROLE ellisp;

\password ellisp;

ALTER ROLE ellisp WITH LOGIN;

CREATE ROLE shiny;

-- no need for a password for shiny, it can only access the db from this machine

CREATE ROLE external\_analyst;

\password external\_analyst;

GRANT ALL PRIVILEGES ON DATABASE twitter TO ellisp;

GRANT ALL PRIVILEGES ON DATABASE survey\_microdata TO ellisp;

\c survey\_microdata;

CREATE SCHEMA nzivs;

CREATE SCHEMA nzis2011;

GRANT ALL PRIVILEGES ON SCHEMA nzivs TO ellisp;

GRANT ALL PRIVILEGES ON SCHEMA nzis2011 TO ellisp;

GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA nzivs TO ellisp;

GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA nzis2011 TO ellisp;

GRANT SELECT ON ALL TABLES IN SCHEMA nzis2011 to external\_analyst;

GRANT SELECT ON ALL TABLES IN SCHEMA nzivs to external\_analyst;

\c twitter

CREATE SCHEMA tweets;

GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA public TO ellisp;

GRANT ALL PRIVILEGES ON ALL TABLES IN SCHEMA tweets TO ellisp;

GRANT SELECT ON ALL TABLES IN SCHEMA tweets TO shiny;

GRANT CONNECT ON DATABASE twitter TO shiny;

\q

We also need to tweak the configuration so the PostgreSQL database is accessible from the outside world (if that’s what we want, which I do).

# follow instructions at https://blog.bigbinary.com/2016/01/23/configure-postgresql-to-allow-remote-connection.html if

# you want to remotely access eg from DBeaver on your laptop. Definitely need a password then

# first add in listen\_addresses = 'localhost' just above the commented out version of #listen\_addresses = 'localhost'

sudo nano /var/lib/pgsql/data/postgresql.conf

# now the client authentication file about how individuals can actually log on.

# Add the below two lines (not the # at beginning) to the bottom of the table.

# lets users log on via password form anywhere. If this doesn't suit your

# risk profile, find something more constrictive...

# host all all 0.0.0.0/0 md5

# host all all ::/0 md5

sudo nano /var/lib/pgsql/data/pg\_hba.conf

sudo systemctl restart postgresql

**A Twitter sample stream database**

**Collecting data**

OK, that wasn’t so bad was it (or was it…). I now have my server running (I stopped it and restarted it as a smaller cheaper instance than the 8GB of RAM I used during that setup) and available to do Useful Stuff. Like collect Twitter data for analysis and dissemination in Shiny:

For a while, I’ve been mildly exercised by the problem of sampling from Twitter.

Code Chunks - Do tweeps with more followers follow tweeps with more followers?

A sampling problem

The first challenge is to get a sample of Twitter users. This is harder than it might seem at first, if the aim is (as it should be) to be representative of the population at large. First challenge is defining that population. Do we mean every Twitter account, every human Twitter account, every account that is used for actively tweeting (why would you restrict it to this, as even reading tweets should surely count?).

There’s no conveniently published population of Twitter users. I’m aware of three broad ways one might go about getting a sample of users:

1. You could do the equivalent of “random digit dialling”, making up numeric Twitter identification numbers and checking them in the Twitter API for existence. This method is in fact what you find if you google “how do I get a sample of Twitter users” but Twitter have made it effectively impossible by the (hidden) way they assign IDs. I observed ID numbers up to 10^17 and as low as 10^6, and sampling random numbers between those extremes hoping to hit one of the 300 million or so actual users sounds like a recipe for getting nowhere.
2. You can pick a node and use a snowball sampling method; that is, follow an edge (either a person the first node follows, or a node who follows them) to another node, record what you need to about that person, then follow another edge to a third node, and so on until you have enough people. This is what I did with the sample labelled “snowball sampling along network”.
3. You can sample a bunch of actual tweets, and treat their authors as your sample. This is what I did with the sample labelled “sample of today’s tweeters”.

Method 1 is I think infeasible. Method 2 will oversample users with lots of followers and who follow lots of people - basically, the more networked you are, the more likely you are to be sampled. On the plus side for Method 2, quiet users who lurk but aren’t tweeting these days will have a chance of selection. Method 3 on the other hand will give a very particular slice of users; but it has the advantage of less obvious dependence between the nodes we pick (all they have in common is tweeting in the last couple of minutes of when we harvest them). I was unsure enough about sampling strategy to try them both.

Here’s code to do the sampling, using Jeff Gentry’s twitteR R package. Note that the code that follows isn’t very robust, and took days to run because of Twitter’s rules on maximum downloads via the public API. There’s lots that can go wrong with grabbing data from the Twitter API, hence the extensive and undisciplined use of try() in the code below in an effort to keep the data harvesting going in the face of various quirks .

First, setup. This requires four different pass phrases which associated with your Twitter account (for obvious reasons, blanked out in the below, which stops this script being fully reproducible as-is).

library(tidyverse)

library(twitteR)

library(GGally)

library(mgcv)

consumer\_key **<-** "XXXXXXX"

consumer\_secret **<-** "XXXXXXXXXX"

access\_token **<-** "XXXXXXXXXX"

access\_secret **<-** "XXXXXXXXXXX"

setup\_twitter\_oauth(consumer\_key, consumer\_secret, access\_token, access\_secret)

As I’m interested in the “average” number of followers of the people a sampled user follows, for each user I sample I’m going to need to find out everyone they follow and estimate how many followers *they* have. This is the thing that takes time. It also exposes a problem; Twitter won’t let you look at more than a certain number of users at once. That number turns out to be 75,000, as I find out from this experiment with Todd Carey:

many\_following\_user **<-** getUser("toddcarey")

many\_ids **<-** many\_following\_user**$**getFriendIDs()

**length**(many\_ids) # returns only 75000 ids; I think the most recent 75,000 he's followed

The problem here is that as Carey follows so many people (1.28 million) it’s infeasible to get the details of them all. In fact, it’s infeasible even just to get all their 1.28 million screen names and then sample from those (I’m very happy to estimate “average number of followers of people X follows” from a sample). I’ll have to deal with getting 75,000 users he follows, then sampling from those 75,000. The big problem here is that I think those are the most recent 75,000 people he’s followed. All else being equal, these are likely to be newer Twitter users than the overall population of people he follows, and hence likely to bias downwards my estimate of the average number of followers *they* have (as newer users will almost certainly have fewer followers on average).

[As an aside, one might wonder what is the point of following 1.28 million people on Twitter; it’s presumably part of a strategy, automated or not, of attracting followers by implicitly agreeing to follow them back. It’s not fully automatic - I established this by following him to see what happened, and nothing did.]

I don’t see what can be done about this, other than note the people who follow more than 75,000 people as potentially suspect in subsequent analysis. There’s not that many of them in my sample anyway.

Anywhere, here’s the code that does the snowball sampling. I start with myself. I take three types of average number of followers of the people X follows: mean (which is highly vulnerable to an arbitrarily large number getting in the sample), 20% trimmed mean and 50% trimmed mean (ie median); but I’m satisfied that 20% trimmed mean is robust and a good measure.

*#=======================snowball sampling method================*

follow\_data **<-** data\_frame(

number\_followers **=** integer(),

number\_following **=** integer(),

mean\_ff **=** numeric(),

trmean\_ff **=** numeric(),

median\_ff **=** numeric(),

screenName **=** character()

)

*# Terminology*

*# A "friend" is someone you follow*

*# A "follower" is someone who follows you*

current\_sn **<-** "ellis2013nz"

set.seed(124)

**for**(i **in** 1**:**1000){

*# save latest copy of results in case of crash*

save(follow\_data, file **=** "follow\_data.rda")

*# number of calls left to make under Twitter's limits*

lims **<-** getCurRateLimitInfo()

x **<-** **as.numeric**(filter(lims, resource **==** "/followers/ids")**$**remaining) **\***

**as.numeric**(filter(lims, resource **==** "/friends/ids")**$**remaining)

**while**(x **==** 0) {

lims **<-** getCurRateLimitInfo()

x **<-** **as.numeric**(filter(lims, resource **==** "/followers/ids")**$**remaining) **\***

**as.numeric**(filter(lims, resource **==** "/friends/ids")**$**remaining)

message("Waiting")

Sys.sleep(60) *# wait 15 minutes (or 1 minute, 15 times)*

}

cat(paste(i, current\_sn, " | "))

current\_user **<-** getUser(current\_sn)

follow\_data[i, "screenName"] **<-** current\_sn

*# who this user follows (what happens if this is eg 1million people? I think it gets truncated at 75,000,*

*# which is not as bad as it sounds as we only need a sample; but the problem will be that this is their*

*# 75,000 most recent people they have followed, who might be biased to be newcomers?):*

*# certain privacy settings stop you seeing someone's friends and followers, in which case*

*# we try to sample another node from the last good person*

all\_my\_friends **<-** character()

try(all\_my\_friends **<-** current\_user**$**getFriendIDs()) *# about 1 second for c.1500 IDs, longer for more*

**if**(**length**(all\_my\_friends) **>** 0){

*# sample some of those friends*

n **<-** **min**(2000, **length**(all\_my\_friends))

sample\_my\_friends **<-** sample(all\_my\_friends, n, replace **=** **FALSE**)

friend\_details **<-** lookupUsers(sample\_my\_friends) *# about 15 seconds*

friend\_details\_df **<-** twListToDF(friend\_details)

follow\_data[i, "number\_followers"] **<-** current\_user**$**followersCount

follow\_data[i, "number\_following"] **<-** current\_user**$**friendsCount

*# mean number of followers of the people this user follows*

follow\_data[i, "mean\_ff"] **<-** mean(friend\_details\_df**$**followersCount)

follow\_data[i, "trmean\_ff"] **<-** mean(friend\_details\_df**$**followersCount, tr **=** 0.2)

follow\_data[i, "median\_ff"] **<-** median(friend\_details\_df**$**followersCount)

}

*# find the next user - flip a coin for it to be either a friend or a follower*

current\_sn **<-** "non\_existent\_user"

**while**(current\_sn **==** "non\_existent\_user"){

**if**(sample(1**:**2, 1) **==** 1){

*# 50% chance*

*# pick one of the user's friends (ie people they follow) at random as the new node to sample*

*# note that as a sampling strategy this will bias us towards people who are followed. People*

*# who have few followers have little chance of being selected here. Does this matter?*

*# Probably not for our main research question, but it does stop us doing meaningful inference*

*# on population totals of number of people followed and number of people following - unless*

*# we could estimate \*how much\* more likely people are to be followed, then we could estimate*

*# weights to make up for it.*

cat("\nsampling a friend\n")

try(current\_sn **<-** sample\_n(friend\_details\_df, 1)**$**screenName)

} **else** {

*# 50% chance*

*# pick one of the user's followers as the new node. This is to mitigate the problem*

*# noted above. We will still end up over-representing people with lots of followers*

*# (who will be sampled by the previuos method), and people who follow lots of people*

*# (more likely to be picked up in this alternative procedure in the next line), but*

*# that is better than \*only\* over sampling people with lots of followers. Probably.*

cat("\nsampling a follower\n")

*# problem here when current\_user has no followers:*

*# a random sample from the last 1000 people to follow this person (ie biased to more*

*# recent, for people with lots of followers, but c'est la vie - important to speed things*

*# up a bit)*

try(follower\_samp **<-** sample(current\_user**$**getFollowerIDs(n **=** 1000), 1))

try(current\_sn **<-** lookupUsers(follower\_samp)[[1]]**$**screenName)

}

}

}

This gets me a sample that looks like the below. Note that this method is prone to sampling individuals more than once, particularly highly networked users (ie lots of followers and/or lots of people following). I’ll deal with that later by taking the average of the estimates of them.

**>** follow\_data **%>%**

**+** arrange(desc(number\_followers))

*# A tibble: 983 x 6*

number\_followers number\_following mean\_ff trmean\_ff median\_ff screenName

**<**dbl**>** **<**dbl**>** **<**dbl**>** **<**dbl**>** **<**dbl**>** **<**chr**>**

1 108538969 203 6002702 1227629 450295 katyperry

2 71131344 1034 3113456 373193 140602 YouTube

3 44212387 183 3208975 594062 255161 BillGates

4 40436829 1868 810532 38519 17959 narendramodi

5 40410386 1867 810495 38519 17926 narendramodi

6 40409932 1867 810489 38519 17926 narendramodi

7 40400380 1868 809897 38479 17924 narendramodi

8 35470473 1661 1559447 334577 219023 SportsCenter

9 24954373 424 1182455 232537 96970 PMOIndia

10 21680806 771 900960 6736 2814 HillaryClinton

# ... with 973 more rows

The snapshot sampling method is a bit simpler. All I need is 1,000 random tweets, which I get by searching for tweets containing the letter “e” (as twitteR doesn’t facilitate a completely open search as far as I can see). This isn’t great - I think it eliminates people using some character sets - but is good enough for a blog.

*#==================snapshot sampling method===========*

*# The first method of sampling will oversample people with lots of followers and people*

*# with lots of friends. As this is potentially linked to the response variable (indirectly)*

*# it might be better to try an alternative sampling method, by grabbing a bunch of tweets.*

*# This method will oversample people who are active tweeting today, which is a different*

*# sort of coverage problem, so still need to be careful.*

follow\_data\_2 **<-** data\_frame(

number\_followers **=** integer(),

number\_following **=** integer(),

mean\_ff **=** numeric(),

trmean\_ff **=** numeric(),

median\_ff **=** numeric(),

screenName **=** character()

)

*# 1000 random tweets with the letter "e" in them:*

tweets **<-** searchTwitter("e", n **=** 1000)

*# the users*

users **<-** unique(sapply(tweets, **function**(x){x**$**screenName}))

**for**(i **in** 1**:length**(users)){

*# save latest copy of results in case of crash*

save(follow\_data\_2, file **=** "follow\_data\_2.rda")

*# number of calls left to make under Twitter's limits*

lims **<-** getCurRateLimitInfo()

x **<-** **as.numeric**(filter(lims, resource **==** "/followers/ids")**$**remaining) **\***

**as.numeric**(filter(lims, resource **==** "/friends/ids")**$**remaining)

**while**(x **==** 0) {

lims **<-** getCurRateLimitInfo()

x **<-** **as.numeric**(filter(lims, resource **==** "/followers/ids")**$**remaining) **\***

**as.numeric**(filter(lims, resource **==** "/friends/ids")**$**remaining)

message("Waiting")

Sys.sleep(60) *# wait 15 minutes (or 1 minute, 15 times)*

}

current\_sn **<-** users[i]

cat(paste(i, current\_sn, " | "))

try({

current\_user **<-** getUser(current\_sn) *# this fails surprisingly often, not sure why*

follow\_data\_2[i, "screenName"] **<-** current\_sn

*# who this user follows (what happens if this is eg 1million people? I think it gets truncated at 15,000,*

*# which is not as bad as it sounds as we only need a sample; but the problem will be that this is their*

*# 15,000 most recent people they have followed, who might be biased to be newcomers?):*

*# certain privacy settings stop you seeing someone's friends and followers, in which case*

*# we try to sample another node from the last good person*

all\_my\_friends **<-** character()

try(all\_my\_friends **<-** current\_user**$**getFriendIDs()) *# about 1 second for c.1500 IDs, longer for more*

**if**(**length**(all\_my\_friends) **>** 0){

*# sample some of those friends*

n **<-** **min**(2000, **length**(all\_my\_friends))

sample\_my\_friends **<-** sample(all\_my\_friends, n, replace **=** **FALSE**)

friend\_details **<-** lookupUsers(sample\_my\_friends) *# about 15 seconds*

friend\_details\_df **<-** twListToDF(friend\_details)

follow\_data\_2[i, "number\_followers"] **<-** current\_user**$**followersCount

follow\_data\_2[i, "number\_following"] **<-** current\_user**$**friendsCount

*# mean number of followers of the people this user follows*

follow\_data\_2[i, "mean\_ff"] **<-** mean(friend\_details\_df**$**followersCount)

follow\_data\_2[i, "trmean\_ff"] **<-** mean(friend\_details\_df**$**followersCount, tr **=** 0.2)

follow\_data\_2[i, "median\_ff"] **<-** median(friend\_details\_df**$**followersCount)

}

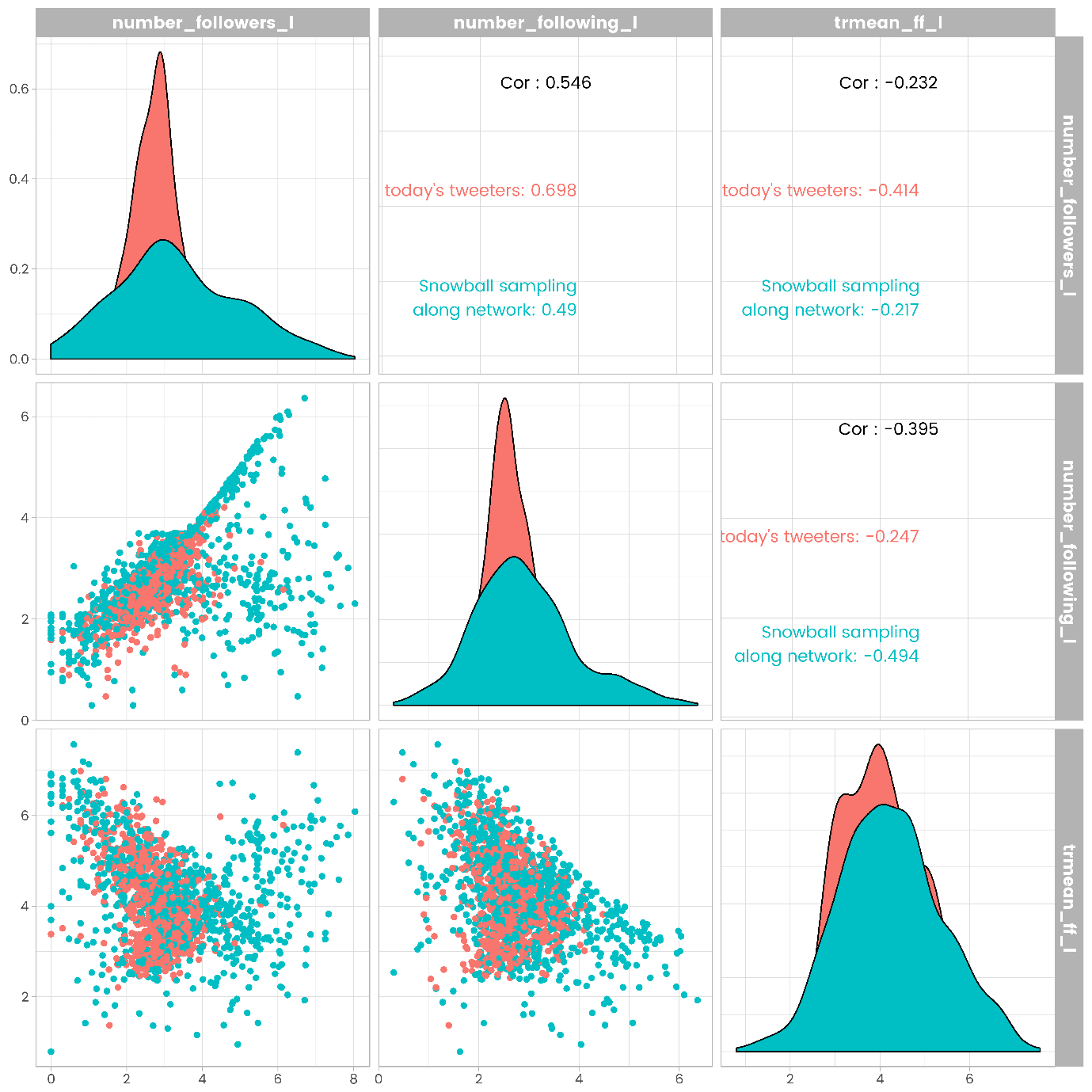
})

}

Results

The resulting numbers are all very skewed distributions. Visually, they look good when you take the logarithm of the original number plus 1 (needed to avoid turning people with 0 friends or 0 followers becoming -Inf). I can justify “+ 1” by saying that, in a way, everyone follows themself and is followed by themselves.

Here’s the distributions and relationships of the key variables, when transformed this way:



We see a strong positive relationship between number of your followers and number of people you are following - until we get to people with many followers (10,000 or more), when the relationship breaks down (visible only in the snowball sample, as the simpler “today’s tweeters” sampling method harvests few such people). Partly this comes from reciprocal follow-back arrangements, partly it is a general indicator of longevity.

We also see a strongly negative relationship between number of people one is following and the average number of their followers. This makes sense and fits in with the notion that if you want to be following people with lots of followers, you have to be quite selective in who you follow, and its best to follow big names in sport, entertainment and perhaps politics. Consider Trump supporter @TheRoyalPosts as an extreme example (not actually included in the final sample). With 41,700 followers of her own, she follows only 120 prominent political accounts, with an astonishing average number of followers themselves of 7 million.

At this point let’s have another look at the graphic I started the post with:

From these last couple of charts, I’m actually pretty happy with both of my sampling methods.

Here’s the code that combines the two samples and produces the graphics above.

*#==================analysis========================*

follow\_data\_proc **<-** follow\_data **%>%**

mutate(sampling\_method **=** "Snowball sampling\nalong network")

follow\_data\_proc\_2 **<-** follow\_data\_2 **%>%**

mutate(sampling\_method **=** "Sample of today's tweeters")

follow\_data\_comb **<-** rbind(follow\_data\_proc, follow\_data\_proc\_2) **%>%**

filter(**!is.na**(trmean\_ff)) **%>%**

group\_by(screenName, sampling\_method) **%>%**

summarise(

number\_followers **=** mean(number\_followers),

number\_following **=** mean(number\_following),

mean\_ff **=** mean(mean\_ff),

trmean\_ff **=** mean(trmean\_ff),

median\_ff **=** mean(median\_ff),

n\_obs **=** n()

) **%>%**

ungroup() **%>%**

mutate(many\_following **=** cut(number\_following, breaks **=** **c**(0, 1000, 5000, 15000, 75000, 10 **^** 9),

labels **=** **c**("0 - 1,000", "1,001 - 5,000", "5,001 - 15,000", "15,001 - 75,000", "75,001 or more")))

follow\_data\_trans **<-** follow\_data\_comb **%>%**

mutate(number\_followers\_l **=** log10(number\_followers **+** 1),

number\_following\_l **=** log10(number\_following **+** 1),

trmean\_ff\_l **=** log10(trmean\_ff **+** 1)) **%>%**

select(number\_followers\_l, number\_following\_l, trmean\_ff\_l, sampling\_method)

ggpairs(follow\_data\_trans, columns **=** 1**:**3, mapping **=** aes(colour **=** sampling\_method))

ggplot(follow\_data\_comb, aes(x **=** number\_followers, y **=** trmean\_ff)) **+**

facet\_wrap(**~**sampling\_method) **+**

geom\_point(aes(fill **=** many\_following), pch **=** 21) **+**

geom\_smooth(method **=** "loess") **+**

scale\_x\_log10("Number of followers, of this person", label **=** comma) **+**

scale\_y\_log10("Trimmed mean number of followers,\nof the people this person follows", label **=** comma) **+**

ggtitle("Do people with more followers follow people with more followers?",

"Apparently 'it depends'.") **+**

scale\_fill\_brewer("Following how\nmany people:", palette **=** "Spectral",

guide **=** guide\_legend(title.hjust **=** 0.5, override.aes **=** **list**(size **=** 5))) **+**

theme(legend.position **=** "right")

Modelling

Finally, I wanted to see if the apparent u-shaped relationship between number of followers and the average number of followers of people one follows was robust to a model that simultaneously modelled the strongly negative relationship between the number of people one follows and that same response variable. It turns out that this is the case. Here are the partial effects of the two variables in a generalized additive model without an interaction term:

And here is how the relationship looks when there is an interaction term. First, as a three-dimensional perspective plot:

… then, more usefully as a heatmap.

The interaction is significant so we’d keep it in even though it complicates the interpretation.

I think the heatmap is the best representation of the data. It shows clearly that the average number of followers of the people one follows is:

* high for people with few followers
* high for people with many followers who *don’t* follow many themselves
* low for people with many followers who *do* follow lots of people themselves

Additionally, there are are very few or no users who follow hundreds of thousands of accounts but have a low number of their own followers (hence the white space in the top left corner).

Here’s the code for the modelling:

mod\_full **<-** gam(trmean\_ff\_l **~** s(number\_followers\_l, number\_following\_l), data **=** follow\_data\_trans)

mod\_simp **<-** gam(trmean\_ff\_l **~** s(number\_followers\_l) **+** s(number\_following\_l), data **=** follow\_data\_trans)

anova(mod\_full, mod\_simp, test **=** "F")

*# simple two variable model*

par(bty **=** "l", font.main **=** 1)

plot(mod\_simp, pages **=** 1, main **=** "No-interaction model")

*# perspective plot*

plot(mod\_full, pages **=** 1, scheme **=** 1, main **=** ("Impact on the trimmed mean number\nof followers of people that are followed"))

*# heat map*

plot(mod\_full, pages **=** 1, scheme **=** 2, hcolors **=** topo.colors(50),

main **=** ("Impact on the trimmed mean number\nof followers of people that are followed"),

xlab **=** "Logarithm of number of followers",

ylab **=** "Logarithm of number people following")

legend("topleft", legend **=** **c**("low", "medium", "high"), fill **=** topo.colors(3), bty **=** "n", ce

On the other hand, Twitter make available several sets of public Tweets that are fully representative:

* The free Sample Tweets API “returns a small random sample of all public Tweets.”
* The Decahose stream provides a 10% sample of all public Tweets
* The Firehose provides all public tweets

The latter two services are for paying customers only. My interest in Twitter is curiousity at most, so I’m only interested in the free sample, which is thought to be around 1% of the Firehose (exactly what proportion it is of the Firehose isn’t publicly known, and is a question of some inferential interest).

So I was interested in the sample stream, but I wanted to collect a sample over a period of time, not just from the day I was going to do some analysis. Even this 1% sample was more than I wanted to pay disk space to store if I were to collect over time, so I decided I would collect 30 seconds of sample streaming data every hour, at a random time within the hour to avoid problems associated with doing the sampling at the same time each day.

I designed a data model to capture the data I was most interested in while discarding attached video and images (this was about saving me disk space; I think serious Twitter analysis would have to do better than just collecting text). It looks like this:

BTW that diagram (and much of the database development) was done with the excellent [universal SQL editor and database admin tool, DBeaver](https://dbeaver.com/). It works with different flavours of relational database and is awesome.

The code that creates and populates that database is available below:

* the SQL that builds the empty database.

Twitter Tables

|  |
| --- |
| CREATE TABLE tweets.batches( |
|  | batch\_id INT PRIMARY KEY, |
|  | collection\_seconds INT, |
|  | time\_collection\_started TIMESTAMP, |
|  | time\_collection\_finished TIMESTAMP, |
|  | tweets\_downloaded INT, |
|  | tweets\_loaded INT, |
|  | retweets\_loaded INT, |
|  | quotes\_loaded INT, |
|  | replies\_loaded INT, |
|  | mentions\_loaded INT, |
|  | hashtags\_loaded INT, |
|  | new\_sources\_loaded INT, |
|  | new\_users\_loaded INT, |
|  | users\_followers\_counted INT, |
|  | time\_load\_completed TIMESTAMP, |
|  | load\_succeeded BOOLEAN |
|  | ); |
|  |  |
|  |  |
|  | CREATE TABLE tweets.sources( |
|  | src\_id INT PRIMARY KEY, |
|  | src\_name TEXT NOT NULL, |
|  | batch\_id INT NOT NULL REFERENCES tweets.batches |
|  |  |
|  | ); |
|  |  |
|  | CREATE TABLE tweets.users( |
|  | user\_id BIGINT PRIMARY KEY, |
|  | screen\_name TEXT NOT NULL, |
|  | account\_created\_at TIMESTAMP, |
|  | batch\_first\_observed INT NOT NULL REFERENCES tweets.batches |
|  | ); |
|  |  |
|  | CREATE TABLE tweets.users\_counts( |
|  | user\_id BIGINT REFERENCES tweets.users, |
|  | followers\_count INT, |
|  | friends\_count INT, |
|  | statuses\_count INT, |
|  | favourites\_count INT, |
|  | batch\_id INT NOT NULL REFERENCES tweets.batches |
|  | ); |
|  | ALTER TABLE tweets.users\_counts ADD PRIMARY KEY (user\_id, batch\_id); |
|  |  |
|  |  |
|  | CREATE TABLE tweets.users\_characteristics( |
|  | user\_id BIGINT NOT NULL REFERENCES tweets.users, |
|  | characteristic TEXT NOT NULL, |
|  | value TEXT NOT NULL, |
|  | batch\_first\_observed INT NOT NULL REFERENCES tweets.batches |
|  | ); |
|  | ALTER TABLE tweets.users\_characteristics ADD PRIMARY KEY (user\_id, characteristic); |
|  |  |
|  |  |
|  | CREATE TABLE tweets.tweets( |
|  | status\_id BIGINT PRIMARY KEY, |
|  | user\_id BIGINT NOT NULL REFERENCES tweets.users, |
|  | text TEXT NOT NULL, |
|  | number\_mentions INT, |
|  | created\_at TIMESTAMP NOT NULL, |
|  | display\_text\_width INT, |
|  | is\_quote BOOLEAN NOT NULL, |
|  | is\_retweet BOOLEAN NOT NULL, |
|  | lang CHAR(3), |
|  | is\_reply BOOLEAN NOT NULL, |
|  | src\_id INT NOT NULL REFERENCES tweets.sources, |
|  | batch\_id INT NOT NULL REFERENCES tweets.batches |
|  | ); |
|  | CREATE INDEX tweeter ON tweets.tweets(user\_id); |
|  |  |
|  | CREATE TABLE tweets.tweets\_rare\_characteristics( |
|  | status\_id BIGINT NOT null references tweets.tweets, |
|  | field TEXT NOT NULL, |
|  | value\_sequence INT NOT NULL, |
|  | value TEXT NOT NULL |
|  | ); |
|  | ALTER TABLE tweets.tweets\_rare\_characteristics ADD PRIMARY KEY (status\_id, field, value\_sequence); |
|  |  |
|  |  |
|  | CREATE TABLE tweets.hashtags( |
|  | status\_id BIGINT REFERENCES tweets.tweets, |
|  | hashtag\_sequence INTEGER, |
|  | hashtag TEXT NOT NULL |
|  | ); |
|  | ALTER TABLE tweets.hashtags ADD PRIMARY KEY (status\_id, hashtag\_sequence); |
|  | CREATE INDEX idx\_hash ON tweets.hashtags (hashtag, status\_id); |
|  |  |
|  | CREATE TABLE tweets.mentions( |
|  | status\_id BIGINT REFERENCES tweets.tweets, |
|  | mentioned\_user\_id BIGINT NOT NULL |
|  | ); |
|  | -- it's probably possible for one 'status' (ie tweet) to mention the same user twice so shouldn't make the combination unique, just make an index: |
|  | CREATE INDEX idx\_men ON tweets.mentions (status\_id, mentioned\_user\_id); |
|  | -- Also, we would have a reference from mentioned\_user\_id to the users table except that we aren't collecting any real information on people |
|  | -- who are just metnioned |
|  |  |
|  |  |
|  | CREATE TABLE tweets.retweeted( |
|  | status\_id BIGINT REFERENCES tweets.tweets, |
|  | retweet\_status\_id BIGINT, |
|  | retweet\_user\_id BIGINT REFERENCES tweets.users |
|  | ); |
|  | ALTER TABLE tweets.retweeted ADD PRIMARY KEY (status\_id, retweet\_status\_id); |
|  |  |
|  | CREATE TABLE tweets.quoted( |
|  | status\_id BIGINT REFERENCES tweets.tweets, |
|  | quoted\_status\_id BIGINT, |
|  | quoted\_user\_id BIGINT REFERENCES tweets.users |
|  | ); |
|  | ALTER TABLE tweets.quoted ADD PRIMARY KEY (status\_id, quoted\_status\_id); |
|  |  |
|  |  |
|  | CREATE TABLE tweets.replies( |
|  | status\_id BIGINT REFERENCES tweets.tweets, |
|  | reply\_to\_status\_id BIGINT, |
|  | reply\_to\_user\_id BIGINT |
|  | ); |
|  | ALTER TABLE tweets.replies ADD PRIMARY KEY (status\_id, reply\_to\_user\_id); |

* the R code that imports a 30 second window of the sample stream and uploads it to the public schema of the twitter database.

Code Chunks for R R code that imports a 30 second window of the sample stream

|  |
| --- |
| # pause for between 0 and 30 minutes, so the time of gathering is random |
|  | Sys.sleep(runif(1, 0, 30 \*60)) |
|  |  |
|  | library(rtweet) |
|  | library(tidyverse) |
|  | library(RPostgres) |
|  |  |
|  | update\_batch <- function(field, value, isstring = FALSE){ |
|  | if(isstring){ |
|  | value <- paste0("'", value, "'") |
|  | } |
|  | sql <- paste0("update tweets.batches set ", field, " = ", value, |
|  | " where batch\_id = ", batch\_id) |
|  | print(sql) |
|  | dbSendQuery(con, sql) |
|  | } |
|  |  |
|  | con <- dbConnect(RPostgres::Postgres(), dbname = "twitter") |
|  |  |
|  | batch\_id <- dbGetQuery(con, "select max(batch\_id) as x from tweets.batches")$x + 1 |
|  | if(is.na(batch\_id)){batch\_id <- 1} |
|  |  |
|  | sql <- paste("insert into tweets.batches(batch\_id) select", batch\_id, " AS batch\_id") |
|  | dbSendQuery(con, sql) |
|  |  |
|  | load("twitter\_token.rda") |
|  |  |
|  | collection\_seconds <- 30 |
|  |  |
|  |  |
|  | update\_batch("collection\_seconds", collection\_seconds) |
|  |  |
|  | batch <- data\_frame(batch\_id = batch\_id, |
|  | collection\_seconds = collection\_seconds, |
|  | time\_collection\_started = Sys.time()) |
|  |  |
|  | systm <- function(){ substring(Sys.time(),1,19)} |
|  |  |
|  | update\_batch("time\_collection\_started", systm(), TRUE) |
|  |  |
|  | st <- stream\_tweets(token = twitter\_token, timeout = collection\_seconds, verbose = FALSE) |
|  |  |
|  | update\_batch("time\_collection\_finished", systm(), TRUE) |
|  |  |
|  | update\_batch("tweets\_downloaded", nrow(st)) |
|  |  |
|  | # caution the 0.6.0 version of rtweet on CRAN imports quite a bit less information than does the 0.6.3 on GitHub |
|  |  |
|  | # Things to think about: |
|  | # \* status\_id is the primary key for tweets |
|  | # \* one user\_id per tweet. An obvious dimension table is user\_id with "latest screen name" and "last observed" |
|  | # columns |
|  | # \* mentions\_user\_id can be NA, a number, or a vector of numbers |
|  | # \* mentions\_screen\_name matches to mentions\_user\_id but I think it cna change |
|  | # \* need a table of user\_id, screen\_name, observation\_time and other things we observed about that user at |
|  | # that time including followers\_count, statuses\_count, favourites\_count, profile\_url, etc, |
|  | # \* a better thought - one user\_slow\_moving with things like screen name and profile; one user\_fast\_moving |
|  | # with things like statustses\_count, favourites\_coutn, that change all the time |
|  | # \* many things are quite sparse eg media, geo\_coords, bbox\_coords |
|  | # \* relatively small number of source (Tweet Deck, Android, etc) - should be coded |
|  |  |
|  | # Tables for: |
|  | # tweets |
|  | # sources |
|  | # mentions |
|  | # hasttags |
|  | # users and their latest screen name |
|  | # users\_slow\_characteristics (long and thin) |
|  | # users\_fast\_characteristics (wide) |
|  | # retweet and quote details |
|  | # tweet locations |
|  |  |
|  |  |
|  | current\_sources <- dbGetQuery(con, "select \* from tweets.sources") |
|  | sourcen <- ifelse(nrow(current\_sources) == 0 , 1, max(current\_sources$src\_id) + 1) |
|  |  |
|  |  |
|  | sources <- data\_frame(src\_name = unique(st$source)) %>% |
|  | left\_join(current\_sources, by = "src\_name") |
|  |  |
|  | new\_sources <- sources %>% |
|  | filter(is.na(src\_id)) |
|  |  |
|  | new\_sources$src\_id <- sourcen:(nrow(new\_sources) - 1 + sourcen) |
|  | new\_sources$batch\_id <- batch\_id |
|  |  |
|  | all\_sources <- rbind(current\_sources, new\_sources) |
|  | rm(sources) |
|  |  |
|  | tweets <- st %>% |
|  | # number of other users mentioned in this tweet: |
|  | mutate(number\_mentions = sapply(mentions\_user\_id, length)) %>% |
|  | select(status\_id, user\_id, text, number\_mentions, source, created\_at, display\_text\_width, |
|  | reply\_to\_status\_id, is\_quote, is\_retweet, lang) %>% |
|  | mutate(is\_reply = !is.na(reply\_to\_status\_id)) %>% |
|  | left\_join(all\_sources, by = c("source" = "src\_name")) %>% |
|  | select(-source, -reply\_to\_status\_id) |
|  |  |
|  | tweets\_rare\_characteristics <- st %>% |
|  | select(status\_id, urls\_url:ext\_media\_type, place\_url:bbox\_coords) %>% |
|  | gather(field, value, -status\_id) %>% |
|  | filter(!is.na(value)) %>% |
|  | group\_by(status\_id, field) %>% |
|  | mutate(value = paste(unlist(lapply(value, c)), collapse="|||")) %>% |
|  | separate(value, sep = "\\|\\|\\|", into = as.character(1:50), fill = "right") %>% |
|  | gather(value\_sequence, value, -status\_id, -field) %>% |
|  | mutate(value\_sequence = as.integer(value\_sequence)) %>% |
|  | filter(!is.na(value) & value != "NA") |
|  |  |
|  | mentions <- st %>% |
|  | select(status\_id, mentions\_user\_id) %>% |
|  | group\_by(status\_id) %>% |
|  | mutate(mentions\_user = paste(unlist(lapply(mentions\_user\_id, c)), collapse=","), |
|  | mentions\_user = ifelse(mentions\_user == "NA", NA, mentions\_user)) %>% |
|  | filter(!is.na(mentions\_user)) %>% |
|  | select(-mentions\_user\_id) %>% |
|  | separate(mentions\_user, sep = ",", into = as.character(1:25), fill = "right") %>% |
|  | gather(mention\_sequence, mentioned\_user\_id, -status\_id) %>% |
|  | filter(!is.na(mentioned\_user\_id)) %>% |
|  | select(-mention\_sequence) |
|  |  |
|  | hashtags <- st %>% |
|  | select(status\_id, hashtags) %>% |
|  | group\_by(status\_id) %>% |
|  | mutate(hash\_string = paste(unlist(lapply(hashtags, c)), collapse=","), |
|  | hash\_string = ifelse(hash\_string == "NA", NA, hash\_string)) %>% |
|  | filter(!is.na(hash\_string)) %>% |
|  | select(-hashtags) %>% |
|  | separate(hash\_string, sep = ",", into = as.character(1:25), fill = "right") %>% |
|  | gather(hashtag\_sequence, hashtag, -status\_id) %>% |
|  | filter(!is.na(hashtag)) %>% |
|  | mutate(hashtag\_sequence = as.integer(hashtag\_sequence)) |
|  |  |
|  |  |
|  | # users' slow characteristics |
|  | tweeters\_slow <- st %>% |
|  | select(user\_id, name, location, description, url, protected, |
|  | verified, |
|  | profile\_url, profile\_expanded\_url, account\_lang, |
|  | profile\_banner\_url, profile\_background\_url, profile\_image\_url) %>% |
|  | distinct() %>% |
|  | gather(characteristic, value, -user\_id) %>% |
|  | filter(!is.na(value)) %>% |
|  | mutate(batch\_first\_observed = batch\_id) |
|  |  |
|  | tweeters\_counts <- st %>% |
|  | select(user\_id, followers\_count, friends\_count, statuses\_count, favourites\_count) %>% |
|  | distinct() |
|  |  |
|  | quoted\_counts <- st %>% |
|  | filter(is\_quote) %>% |
|  | select(quoted\_user\_id, quoted\_followers\_count, quoted\_friends\_count, quoted\_statuses\_count, |
|  | quoted\_favorite\_count) %>% |
|  | rename( |
|  | user\_id = quoted\_user\_id, |
|  | followers\_count = quoted\_followers\_count, |
|  | friends\_count = quoted\_friends\_count, |
|  | favourites\_count = quoted\_favorite\_count, |
|  | statuses\_count = quoted\_statuses\_count) |
|  |  |
|  | retweet\_counts <- st %>% |
|  | filter(is\_retweet) %>% |
|  | select(retweet\_user\_id, retweet\_followers\_count, retweet\_friends\_count, retweet\_statuses\_count, |
|  | retweet\_favorite\_count) %>% |
|  | rename( |
|  | user\_id = retweet\_user\_id, |
|  | followers\_count = retweet\_followers\_count, |
|  | favourites\_count = retweet\_favorite\_count, |
|  | friends\_count = retweet\_friends\_count, |
|  | statuses\_count = retweet\_statuses\_count) |
|  |  |
|  |  |
|  | users1 <- st %>% |
|  | select(user\_id, screen\_name, account\_created\_at) %>% |
|  | distinct(user\_id, screen\_name, .keep\_all = TRUE) |
|  |  |
|  | users2 <- st %>% |
|  | filter(is\_quote) %>% |
|  | select(quoted\_user\_id, quoted\_screen\_name) %>% |
|  | rename(user\_id = quoted\_user\_id, |
|  | screen\_name = quoted\_screen\_name) %>% |
|  | mutate(account\_created\_at = NA) %>% |
|  | filter(!user\_id %in% users1$user\_id) %>% |
|  | distinct(user\_id, screen\_name, .keep\_all = TRUE) |
|  |  |
|  | users3 <- st %>% |
|  | filter(is\_retweet) %>% |
|  | select(retweet\_user\_id, retweet\_screen\_name) %>% |
|  | rename(user\_id = retweet\_user\_id, |
|  | screen\_name = retweet\_screen\_name) %>% |
|  | mutate(account\_created\_at = NA) %>% |
|  | filter(!user\_id %in% c(users1$user\_id, users2$user\_id)) %>% |
|  | distinct(user\_id, screen\_name, .keep\_all = TRUE) |
|  |  |
|  |  |
|  | users <- rbind(users1, users2, users3) %>% |
|  | mutate\_("batch\_id" = batch\_id) |
|  |  |
|  | users\_counts <- rbind(tweeters\_counts, retweet\_counts, quoted\_counts) %>% |
|  | distinct(user\_id, observed\_at, .keep\_all = TRUE) %>% |
|  | mutate\_("batch\_id" = batch\_id) |
|  |  |
|  | retweeted <- st %>% |
|  | filter(is\_retweet) %>% |
|  | select(status\_id, retweet\_status\_id, retweet\_user\_id) |
|  |  |
|  | quoted <- st %>% |
|  | filter(is\_quote) %>% |
|  | select(status\_id, quoted\_status\_id, quoted\_user\_id) |
|  |  |
|  | replies <- st %>% |
|  | filter(!is.na(reply\_to\_status\_id)) %>% |
|  | select(status\_id, reply\_to\_status\_id, reply\_to\_user\_id) |
|  |  |
|  | #========================write to staging schema (public) in db================= |
|  |  |
|  |  |
|  | dbWriteTable(con, "sources", new\_sources, row.names = FALSE, overwrite= TRUE) |
|  | update\_batch("new\_sources\_loaded", nrow(new\_sources)) |
|  |  |
|  | dbWriteTable(con, "users", users, row.names = FALSE, overwrite = TRUE) |
|  |  |
|  | dbWriteTable(con, "users\_counts", users\_counts, row.names = FALSE, overwrite = TRUE) |
|  | update\_batch("users\_followers\_counted", nrow(users\_counts)) |
|  |  |
|  | dbWriteTable(con, "users\_characteristics", tweeters\_slow, row.names = FALSE, overwrite = TRUE) |
|  |  |
|  | dbWriteTable(con, "retweeted", retweeted, row.names = FALSE, overwrite= TRUE) |
|  | update\_batch("retweets\_loaded", nrow(retweeted)) |
|  |  |
|  | dbWriteTable(con, "quoted", quoted, row.names = FALSE, overwrite= TRUE) |
|  | update\_batch("quotes\_loaded", nrow(quoted)) |
|  |  |
|  |  |
|  | dbWriteTable(con, "tweets", tweets, row.names = FALSE, overwrite= TRUE) |
|  | update\_batch("tweets\_loaded", nrow(tweets)) |
|  |  |
|  | dbWriteTable(con, "tweets\_rare\_characteristics", tweets\_rare\_characteristics, |
|  | row.names = FALSE, overwrite = TRUE) |
|  |  |
|  |  |
|  | dbWriteTable(con, "mentions", mentions, row.names = FALSE, overwrite= TRUE) |
|  | update\_batch("mentions\_loaded", nrow(mentions)) |
|  |  |
|  |  |
|  | dbWriteTable(con, "hashtags", hashtags, row.names = FALSE, overwrite= TRUE) |
|  | update\_batch("hashtags\_loaded", nrow(hashtags)) |
|  |  |
|  |  |
|  | dbWriteTable(con, "replies", replies, row.names = FALSE, overwrite= TRUE) |
|  | update\_batch("replies\_loaded", nrow(replies)) |
|  |  |
|  | # run the SQL ETL here. Need a way to stop if an error occurs |
|  | system("psql -d twitter --file=gather-data/etl.sql") |
|  |  |
|  | update\_batch("time\_load\_completed", systm(), TRUE) |
|  | update\_batch("load\_succeeded", TRUE) |
|  |  |
|  | dbDisconnect(con) |

* the SQL that transforms and loads the data into the twitter.tweets schema. This does the work, for example, of matching users in the latest sample with previously observed users; making sure that re-tweeted users are in the users table even if we haven’t seen them directly tweet something themselves; and so on.

|  |
| --- |
| /\* |
|  | This script is for moving the data in tables in the public schema, where they have been dumped by R, to the tweets schema |
|  |  |
|  | \*/ |
|  |  |
|  | -------------------------------Sources-------------- |
|  | -- These need to be added first |
|  | INSERT INTO tweets.sources(src\_id, src\_name, batch\_id) |
|  | SELECT |
|  | CAST(src\_id AS INT), |
|  | src\_name, |
|  | batch\_id |
|  | FROM public.sources; |
|  |  |
|  |  |
|  |  |
|  | ----------------------------Users------------------------------ |
|  | -- will first need to reduce public.users to just those rows not already in there |
|  | -- (or create a temporary table that does it, probably easier than DELETE) |
|  | --DROP TABLE IF EXISTS public.users2; |
|  | --DROP TABLE IF EXISTS public.users3; |
|  |  |
|  | SELECT |
|  | CAST(user\_id AS BIGINT) AS user\_id, |
|  | screen\_name, account\_created\_at, batch\_id |
|  | INTO public.users2 |
|  | FROM public.users; |
|  |  |
|  | SELECT a.\* |
|  | INTO public.users3 |
|  | FROM public.users2 AS a |
|  | LEFT JOIN tweets.users AS b |
|  | ON a.user\_id = b.user\_id |
|  | WHERE b.user\_id IS NULL; |
|  |  |
|  |  |
|  |  |
|  | INSERT INTO tweets.users(user\_id, screen\_name, account\_created\_at, batch\_first\_observed) |
|  | SELECT |
|  | CAST(user\_id AS BIGINT), |
|  | screen\_name, |
|  | account\_created\_at, |
|  | batch\_id |
|  | FROM public.users3; |
|  |  |
|  | UPDATE tweets.batches |
|  | SET new\_users\_loaded = (SELECT COUNT(1) FROM public.users3) |
|  | WHERE batch\_id = (SELECT batch\_id FROM public.users LIMIT 1); |
|  |  |
|  |  |
|  | INSERT INTO tweets.users\_counts(user\_id, followers\_count, friends\_count, statuses\_count, favourites\_count, batch\_id) |
|  | SELECT |
|  | CAST(user\_id AS BIGINT), |
|  | followers\_count, |
|  | friends\_count, |
|  | statuses\_count, |
|  | favourites\_count, |
|  | batch\_id |
|  | FROM public.users\_counts; |
|  |  |
|  |  |
|  | INSERT INTO tweets.users\_characteristics(user\_id, characteristic, value, batch\_first\_observed) |
|  | SELECT |
|  | CAST(a.user\_id AS BIGINT), |
|  | a.characteristic, |
|  | a.value, |
|  | a.batch\_first\_observed |
|  | FROM public.users\_characteristics AS a |
|  | INNER JOIN public.users3 AS b |
|  | ON CAST(a.user\_id AS BIGINT) = b.user\_id; |
|  |  |
|  |  |
|  | DROP TABLE public.users2; |
|  | DROP TABLE public.users3; |
|  |  |
|  | -----------------------Tweets------------------------- |
|  | INSERT INTO tweets.tweets( |
|  | status\_id, |
|  | user\_id, |
|  | text, |
|  | number\_mentions, |
|  | created\_at, |
|  | display\_text\_width, |
|  | is\_quote, |
|  | is\_retweet, |
|  | lang, |
|  | is\_reply, |
|  | src\_id, |
|  | batch\_id) |
|  | SELECT |
|  | CAST(status\_id AS BIGINT), |
|  | CAST(user\_id AS BIGINT), |
|  | text, |
|  | number\_mentions, |
|  | created\_at, |
|  | display\_text\_width, |
|  | is\_quote, |
|  | is\_retweet, |
|  | lang, |
|  | is\_reply, |
|  | src\_id, |
|  | batch\_id |
|  | FROM public.tweets; |
|  |  |
|  |  |
|  | INSERT INTO tweets.tweets\_rare\_characteristics(status\_id, field, value\_sequence, value) |
|  | SELECT |
|  | CAST(status\_id AS BIGINT), |
|  | field, |
|  | value\_sequence, |
|  | VALUE |
|  | FROM public.tweets\_rare\_characteristics; |
|  |  |
|  |  |
|  | ---------------------------Mentions and hashtags------------------------ |
|  | INSERT INTO tweets.mentions(status\_id, mentioned\_user\_id) |
|  | SELECT |
|  | CAST(status\_id AS BIGINT), |
|  | CAST(mentioned\_user\_id AS BIGINT) |
|  | FROM public.mentions; |
|  |  |
|  |  |
|  |  |
|  | INSERT INTO tweets.hashtags(status\_id, hashtag\_sequence, hashtag) |
|  | SELECT |
|  | CAST(status\_id AS BIGINT), |
|  | hashtag\_sequence, |
|  | hashtag |
|  | FROM public.hashtags; |
|  |  |
|  | -------------------------------retweets, quoted and replies-------------------------- |
|  | INSERT INTO tweets.retweeted(status\_id, retweet\_status\_id, retweet\_user\_id) |
|  | SELECT |
|  | CAST(status\_id AS BIGINT), |
|  | CAST(retweet\_status\_id AS BIGINT), |
|  | CAST(retweet\_user\_id AS BIGINT) |
|  | FROM public.retweeted; |
|  |  |
|  | INSERT INTO tweets.quoted(status\_id, quoted\_status\_id, quoted\_user\_id) |
|  | SELECT |
|  | CAST(status\_id AS BIGINT), |
|  | CAST(quoted\_status\_id AS BIGINT), |
|  | CAST(quoted\_user\_id AS BIGINT) |
|  | FROM public.quoted; |
|  |  |
|  |  |
|  | INSERT INTO tweets.replies(status\_id, reply\_to\_status\_id, reply\_to\_user\_id) |
|  | SELECT |
|  | CAST(status\_id AS BIGINT), |
|  | CAST(reply\_to\_status\_id AS BIGINT), |
|  | CAST(reply\_to\_user\_id AS BIGINT) |
|  | FROM public.replies; |
|  |  |
|  |  |
|  | DROP TABLE public.sources; |
|  | DROP TABLE public.users; |
|  | DROP TABLE public.users\_counts; |
|  | DROP TABLE public.users\_characteristics; |
|  | DROP TABLE public.retweeted; |
|  | DROP TABLE public.quoted; |
|  | DROP TABLE public.tweets; |
|  | DROP TABLE public.mentions; |
|  | DROP TABLE public.hashtags; |
|  | DROP TABLE public.replies; |
|  | DROP TABLE public.tweets\_rare\_characteristics; |

* the shell script that is activated by a cron job 24 times a day and runs the above R and SQL.

Code Chunks – import\_stream.R

|  |
| --- |
| # pause for between 0 and 30 minutes, so the time of gathering is random |
|  | Sys.sleep(runif(1, 0, 30 \*60)) | |
|  |  | |
|  | library(rtweet) | |
|  | library(tidyverse) | |
|  | library(RPostgres) | |
|  |  | |
|  | update\_batch <- function(field, value, isstring = FALSE){ | |
|  | if(isstring){ | |
|  | value <- paste0("'", value, "'") | |
|  | } | |
|  | sql <- paste0("update tweets.batches set ", field, " = ", value, | |
|  | " where batch\_id = ", batch\_id) | |
|  | print(sql) | |
|  | dbSendQuery(con, sql) | |
|  | } | |
|  |  | |
|  | con <- dbConnect(RPostgres::Postgres(), dbname = "twitter") | |
|  |  | |
|  | batch\_id <- dbGetQuery(con, "select max(batch\_id) as x from tweets.batches")$x + 1 | |
|  | if(is.na(batch\_id)){batch\_id <- 1} | |
|  |  | |
|  | sql <- paste("insert into tweets.batches(batch\_id) select", batch\_id, " AS batch\_id") | |
|  | dbSendQuery(con, sql) | |
|  |  | |
|  | load("twitter\_token.rda") | |
|  |  | |
|  | collection\_seconds <- 30 | |
|  |  | |
|  |  | |
|  | update\_batch("collection\_seconds", collection\_seconds) | |
|  |  | |
|  | batch <- data\_frame(batch\_id = batch\_id, | |
|  | collection\_seconds = collection\_seconds, | |
|  | time\_collection\_started = Sys.time()) | |
|  |  | |
|  | systm <- function(){ substring(Sys.time(),1,19)} | |
|  |  | |
|  | update\_batch("time\_collection\_started", systm(), TRUE) | |
|  |  | |
|  | st <- stream\_tweets(token = twitter\_token, timeout = collection\_seconds, verbose = FALSE) | |
|  |  | |
|  | update\_batch("time\_collection\_finished", systm(), TRUE) | |
|  |  | |
|  | update\_batch("tweets\_downloaded", nrow(st)) | |
|  |  | |
|  | # caution the 0.6.0 version of rtweet on CRAN imports quite a bit less information than does the 0.6.3 on GitHub | |
|  |  | |
|  | # Things to think about: | |
|  | # \* status\_id is the primary key for tweets | |
|  | # \* one user\_id per tweet. An obvious dimension table is user\_id with "latest screen name" and "last observed" | |
|  | # columns | |
|  | # \* mentions\_user\_id can be NA, a number, or a vector of numbers | |
|  | # \* mentions\_screen\_name matches to mentions\_user\_id but I think it cna change | |
|  | # \* need a table of user\_id, screen\_name, observation\_time and other things we observed about that user at | |
|  | # that time including followers\_count, statuses\_count, favourites\_count, profile\_url, etc, | |
|  | # \* a better thought - one user\_slow\_moving with things like screen name and profile; one user\_fast\_moving | |
|  | # with things like statustses\_count, favourites\_coutn, that change all the time | |
|  | # \* many things are quite sparse eg media, geo\_coords, bbox\_coords | |
|  | # \* relatively small number of source (Tweet Deck, Android, etc) - should be coded | |
|  |  | |
|  | # Tables for: | |
|  | # tweets | |
|  | # sources | |
|  | # mentions | |
|  | # hasttags | |
|  | # users and their latest screen name | |
|  | # users\_slow\_characteristics (long and thin) | |
|  | # users\_fast\_characteristics (wide) | |
|  | # retweet and quote details | |
|  | # tweet locations | |
|  |  | |
|  |  | |
|  | current\_sources <- dbGetQuery(con, "select \* from tweets.sources") | |
|  | sourcen <- ifelse(nrow(current\_sources) == 0 , 1, max(current\_sources$src\_id) + 1) | |
|  |  | |
|  |  | |
|  | sources <- data\_frame(src\_name = unique(st$source)) %>% | |
|  | left\_join(current\_sources, by = "src\_name") | |
|  |  | |
|  | new\_sources <- sources %>% | |
|  | filter(is.na(src\_id)) | |
|  |  | |
|  | new\_sources$src\_id <- sourcen:(nrow(new\_sources) - 1 + sourcen) | |
|  | new\_sources$batch\_id <- batch\_id | |
|  |  | |
|  | all\_sources <- rbind(current\_sources, new\_sources) | |
|  | rm(sources) | |
|  |  | |
|  | tweets <- st %>% | |
|  | # number of other users mentioned in this tweet: | |
|  | mutate(number\_mentions = sapply(mentions\_user\_id, length)) %>% | |
|  | select(status\_id, user\_id, text, number\_mentions, source, created\_at, display\_text\_width, | |
|  | reply\_to\_status\_id, is\_quote, is\_retweet, lang) %>% | |
|  | mutate(is\_reply = !is.na(reply\_to\_status\_id)) %>% | |
|  | left\_join(all\_sources, by = c("source" = "src\_name")) %>% | |
|  | select(-source, -reply\_to\_status\_id) | |
|  |  | |
|  | tweets\_rare\_characteristics <- st %>% | |
|  | select(status\_id, urls\_url:ext\_media\_type, place\_url:bbox\_coords) %>% | |
|  | gather(field, value, -status\_id) %>% | |
|  | filter(!is.na(value)) %>% | |
|  | group\_by(status\_id, field) %>% | |
|  | mutate(value = paste(unlist(lapply(value, c)), collapse="|||")) %>% | |
|  | separate(value, sep = "\\|\\|\\|", into = as.character(1:50), fill = "right") %>% | |
|  | gather(value\_sequence, value, -status\_id, -field) %>% | |
|  | mutate(value\_sequence = as.integer(value\_sequence)) %>% | |
|  | filter(!is.na(value) & value != "NA") | |
|  |  | |
|  | mentions <- st %>% | |
|  | select(status\_id, mentions\_user\_id) %>% | |
|  | group\_by(status\_id) %>% | |
|  | mutate(mentions\_user = paste(unlist(lapply(mentions\_user\_id, c)), collapse=","), | |
|  | mentions\_user = ifelse(mentions\_user == "NA", NA, mentions\_user)) %>% | |
|  | filter(!is.na(mentions\_user)) %>% | |
|  | select(-mentions\_user\_id) %>% | |
|  | separate(mentions\_user, sep = ",", into = as.character(1:25), fill = "right") %>% | |
|  | gather(mention\_sequence, mentioned\_user\_id, -status\_id) %>% | |
|  | filter(!is.na(mentioned\_user\_id)) %>% | |
|  | select(-mention\_sequence) | |
|  |  | |
|  | hashtags <- st %>% | |
|  | select(status\_id, hashtags) %>% | |
|  | group\_by(status\_id) %>% | |
|  | mutate(hash\_string = paste(unlist(lapply(hashtags, c)), collapse=","), | |
|  | hash\_string = ifelse(hash\_string == "NA", NA, hash\_string)) %>% | |
|  | filter(!is.na(hash\_string)) %>% | |
|  | select(-hashtags) %>% | |
|  | separate(hash\_string, sep = ",", into = as.character(1:25), fill = "right") %>% | |
|  | gather(hashtag\_sequence, hashtag, -status\_id) %>% | |
|  | filter(!is.na(hashtag)) %>% | |
|  | mutate(hashtag\_sequence = as.integer(hashtag\_sequence)) | |
|  |  | |
|  |  | |
|  | # users' slow characteristics | |
|  | tweeters\_slow <- st %>% | |
|  | select(user\_id, name, location, description, url, protected, | |
|  | verified, | |
|  | profile\_url, profile\_expanded\_url, account\_lang, | |
|  | profile\_banner\_url, profile\_background\_url, profile\_image\_url) %>% | |
|  | distinct() %>% | |
|  | gather(characteristic, value, -user\_id) %>% | |
|  | filter(!is.na(value)) %>% | |
|  | mutate(batch\_first\_observed = batch\_id) | |
|  |  | |
|  | tweeters\_counts <- st %>% | |
|  | select(user\_id, followers\_count, friends\_count, statuses\_count, favourites\_count) %>% | |
|  | distinct() | |
|  |  | |
|  | quoted\_counts <- st %>% | |
|  | filter(is\_quote) %>% | |
|  | select(quoted\_user\_id, quoted\_followers\_count, quoted\_friends\_count, quoted\_statuses\_count, | |
|  | quoted\_favorite\_count) %>% | |
|  | rename( | |
|  | user\_id = quoted\_user\_id, | |
|  | followers\_count = quoted\_followers\_count, | |
|  | friends\_count = quoted\_friends\_count, | |
|  | favourites\_count = quoted\_favorite\_count, | |
|  | statuses\_count = quoted\_statuses\_count) | |
|  |  | |
|  | retweet\_counts <- st %>% | |
|  | filter(is\_retweet) %>% | |
|  | select(retweet\_user\_id, retweet\_followers\_count, retweet\_friends\_count, retweet\_statuses\_count, | |
|  | retweet\_favorite\_count) %>% | |
|  | rename( | |
|  | user\_id = retweet\_user\_id, | |
|  | followers\_count = retweet\_followers\_count, | |
|  | favourites\_count = retweet\_favorite\_count, | |
|  | friends\_count = retweet\_friends\_count, | |
|  | statuses\_count = retweet\_statuses\_count) | |
|  |  | |
|  |  | |
|  | users1 <- st %>% | |
|  | select(user\_id, screen\_name, account\_created\_at) %>% | |
|  | distinct(user\_id, screen\_name, .keep\_all = TRUE) | |
|  |  | |
|  | users2 <- st %>% | |
|  | filter(is\_quote) %>% | |
|  | select(quoted\_user\_id, quoted\_screen\_name) %>% | |
|  | rename(user\_id = quoted\_user\_id, | |
|  | screen\_name = quoted\_screen\_name) %>% | |
|  | mutate(account\_created\_at = NA) %>% | |
|  | filter(!user\_id %in% users1$user\_id) %>% | |
|  | distinct(user\_id, screen\_name, .keep\_all = TRUE) | |
|  |  | |
|  | users3 <- st %>% | |
|  | filter(is\_retweet) %>% | |
|  | select(retweet\_user\_id, retweet\_screen\_name) %>% | |
|  | rename(user\_id = retweet\_user\_id, | |
|  | screen\_name = retweet\_screen\_name) %>% | |
|  | mutate(account\_created\_at = NA) %>% | |
|  | filter(!user\_id %in% c(users1$user\_id, users2$user\_id)) %>% | |
|  | distinct(user\_id, screen\_name, .keep\_all = TRUE) | |
|  |  | |
|  |  | |
|  | users <- rbind(users1, users2, users3) %>% | |
|  | mutate\_("batch\_id" = batch\_id) | |
|  |  | |
|  | users\_counts <- rbind(tweeters\_counts, retweet\_counts, quoted\_counts) %>% | |
|  | distinct(user\_id, observed\_at, .keep\_all = TRUE) %>% | |
|  | mutate\_("batch\_id" = batch\_id) | |
|  |  | |
|  | retweeted <- st %>% | |
|  | filter(is\_retweet) %>% | |
|  | select(status\_id, retweet\_status\_id, retweet\_user\_id) | |
|  |  | |
|  | quoted <- st %>% | |
|  | filter(is\_quote) %>% | |
|  | select(status\_id, quoted\_status\_id, quoted\_user\_id) | |
|  |  | |
|  | replies <- st %>% | |
|  | filter(!is.na(reply\_to\_status\_id)) %>% | |
|  | select(status\_id, reply\_to\_status\_id, reply\_to\_user\_id) | |
|  |  | |
|  | #========================write to staging schema (public) in db================= | |
|  |  | |
|  |  | |
|  | dbWriteTable(con, "sources", new\_sources, row.names = FALSE, overwrite= TRUE) | |
|  | update\_batch("new\_sources\_loaded", nrow(new\_sources)) | |
|  |  | |
|  | dbWriteTable(con, "users", users, row.names = FALSE, overwrite = TRUE) | |
|  |  | |
|  | dbWriteTable(con, "users\_counts", users\_counts, row.names = FALSE, overwrite = TRUE) | |
|  | update\_batch("users\_followers\_counted", nrow(users\_counts)) | |
|  |  | |
|  | dbWriteTable(con, "users\_characteristics", tweeters\_slow, row.names = FALSE, overwrite = TRUE) | |
|  |  | |
|  | dbWriteTable(con, "retweeted", retweeted, row.names = FALSE, overwrite= TRUE) | |
|  | update\_batch("retweets\_loaded", nrow(retweeted)) | |
|  |  | |
|  | dbWriteTable(con, "quoted", quoted, row.names = FALSE, overwrite= TRUE) | |
|  | update\_batch("quotes\_loaded", nrow(quoted)) | |
|  |  | |
|  |  | |
|  | dbWriteTable(con, "tweets", tweets, row.names = FALSE, overwrite= TRUE) | |
|  | update\_batch("tweets\_loaded", nrow(tweets)) | |
|  |  | |
|  | dbWriteTable(con, "tweets\_rare\_characteristics", tweets\_rare\_characteristics, | |
|  | row.names = FALSE, overwrite = TRUE) | |
|  |  | |
|  |  | |
|  | dbWriteTable(con, "mentions", mentions, row.names = FALSE, overwrite= TRUE) | |
|  | update\_batch("mentions\_loaded", nrow(mentions)) | |
|  |  | |
|  |  | |
|  | dbWriteTable(con, "hashtags", hashtags, row.names = FALSE, overwrite= TRUE) | |
|  | update\_batch("hashtags\_loaded", nrow(hashtags)) | |
|  |  | |
|  |  | |
|  | dbWriteTable(con, "replies", replies, row.names = FALSE, overwrite= TRUE) | |
|  | update\_batch("replies\_loaded", nrow(replies)) | |
|  |  | |
|  | # run the SQL ETL here. Need a way to stop if an error occurs | |
|  | system("psql -d twitter --file=gather-data/etl.sql") | |
|  |  | |
|  | update\_batch("time\_load\_completed", systm(), TRUE) | |
|  | update\_batch("load\_succeeded", TRUE) | |
|  |  | |
|  | dbDisconnect(con)  Shell Script | |
| #!/bin/bash | |
|  | | # don't forget to do chmod -x import-stream.sh |
|  | | cd ~/twitter-misc/ |
|  | | Rscript gather-data/import-stream.R > gather-data/twitter\_log.txt |

This has now been running smoothly since 17 May 2018, apart from one day last week when I botched an R upgrade and it all went down for half a day before I noticed (lesson learned – run update.package(ask = FALSE, checkBuilt = TRUE) to ensure your R packages all keep working after the R internals change). So far the database is about 3GB in size, and I’m quite happy to let it grow quite a bit more than that.

**What do we find out?**

So far the main use I’ve put this data to is the Shiny app that I’ve scattered a few screenshots of in this blog post. That Shiny app writes its own SQL based on the inputs provided by the user (eg date range), queries the database and produces charts.

Shiny App Code – Server.R

|  |
| --- |
| library(shiny) |
|  | library(ggplot2) |
|  | library(dplyr) |
|  | library(tidyr) |
|  | library(scales) |
|  | library(wordcloud) |
|  | library(RPostgres) |
|  | library(viridis) |
|  | library(stringr) |
|  | library(forcats) |
|  | library(ggseas) |
|  | library(tools) |
|  |  |
|  | theme\_set(theme\_dark(base\_family = "FreeSans")) |
|  |  |
|  | res <- 72 |
|  |  |
|  | con <- dbConnect(RPostgres::Postgres(), dbname = "twitter") |
|  |  |
|  |  |
|  |  |
|  | tweeters\_sql <- paste(readLines("sql/tweeters.sql"), collapse = "\n") |
|  | hash\_sql <- paste(readLines("sql/hash.sql"), collapse = "\n") |
|  | batches\_sql <- paste(readLines("sql/batches.sql"), collapse = "\n") |
|  | retweets\_sql <- paste(readLines("sql/popular\_text.sql"), collapse = "\n") |
|  |  |
|  | shinyServer(function(input, output, session) { |
|  |  |
|  | #-------------dynamicall change SQL according to inputs------------ |
|  | the\_hash\_sql <- reactive({ |
|  | tmp <- gsub("the\_date\_1", input$date[1], hash\_sql) |
|  | tmp <- gsub("the\_date\_2", input$date[2], tmp) |
|  | return(tmp) |
|  | }) |
|  |  |
|  | the\_tweeters\_sql <- reactive({ |
|  | tmp <- gsub("the\_date\_1", input$date[1], tweeters\_sql) |
|  | tmp <- gsub("the\_date\_2", input$date[2], tmp) |
|  | return(tmp) |
|  | }) |
|  |  |
|  | the\_batches\_sql <- reactive({ |
|  | tmp <- gsub("the\_date\_1", input$date[1], batches\_sql) |
|  | tmp <- gsub("the\_date\_2", input$date[2], tmp) |
|  | return(tmp) |
|  | }) |
|  |  |
|  | the\_retweets\_sql <- reactive({ |
|  | tmp <- gsub("the\_date\_1", input$date[1], retweets\_sql) |
|  | tmp <- gsub("the\_date\_2", input$date[2], tmp) |
|  | return(tmp) |
|  | }) |
|  |  |
|  | #-----------------download data---------------- |
|  | tweeters <- reactive({ |
|  | dbGetQuery(con, the\_tweeters\_sql(), stringsAsFactors = FALSE) %>% |
|  | as\_tibble() |
|  | }) |
|  |  |
|  | batches <- reactive({ |
|  | dbGetQuery(con, the\_batches\_sql(), stringsAsFactors = FALSE) %>% |
|  | as\_tibble() |
|  | }) |
|  |  |
|  |  |
|  | hashtags <- reactive({ |
|  | dbGetQuery(con, the\_hash\_sql(), stringsAsFactors = FALSE) %>% |
|  | as\_tibble() %>% |
|  | mutate(lang = str\_trim(lang)) |
|  | }) |
|  |  |
|  | retweets <- reactive({ |
|  | dbGetQuery(con, the\_retweets\_sql(), stringsAsFactors = FALSE) %>% |
|  | as\_tibble() %>% |
|  | mutate(lang = str\_trim(lang)) |
|  | }) |
|  |  |
|  |  |
|  | #------------more data processing------------- |
|  | hash\_data\_full <- reactive({ |
|  | hashtags() %>% |
|  | filter(lang %in% input$langs) %>% |
|  | group\_by(hashtag) %>% |
|  | summarise(freq = sum(freq)) %>% |
|  | arrange(desc(freq)) |
|  | }) |
|  |  |
|  | hash\_data <- reactive({ |
|  | hash\_data\_full() %>% |
|  | slice(1:80) |
|  | }) |
|  |  |
|  | retweets\_data <- reactive({ |
|  | tmp <- retweets() %>% |
|  | filter(lang %in% input$langs) %>% |
|  | select(-lang, -rank) %>% |
|  | arrange(desc(observed\_retweets)) %>% |
|  | mutate(text = gsub("&amp;", "&", text, fixed = TRUE)) |
|  | names(tmp) <- toTitleCase(gsub("\_", " ", names(tmp))) |
|  | return(tmp) |
|  | }) |
|  |  |
|  | #----------------define graphics------------------ |
|  | tweeters\_plot <- reactive({ |
|  | p <- tweeters() %>% |
|  | slice(1:20) %>% |
|  | mutate(freq = as.numeric(freq), |
|  | screen\_name = fct\_reorder(screen\_name, freq)) %>% |
|  | ggplot(aes(y = screen\_name, x = freq, label = round(inv\_prop, -1))) + |
|  | geom\_text(colour = "yellow", family = "FreeSans") + |
|  | labs(x = "Count in sample", y = "") + |
|  | ggtitle(paste("Prolific tweeters from", input$date[1], |
|  | "to", input$date[2]), |
|  | "Numbers on graphic show what proportion (eg 1 in 10,000) of all tweets are from this person ") |
|  | return(p) |
|  | }) |
|  |  |
|  | batches\_plot <- reactive({ |
|  | p <- batches() %>% |
|  | select(time\_collection\_started, tweets\_loaded:users\_followers\_counted) %>% |
|  | gather(variable, value, -time\_collection\_started) %>% |
|  | mutate(variable = gsub("\_loaded$", "", variable), |
|  | variable = gsub("\_", " ", variable)) %>% |
|  | ggplot(aes(x = time\_collection\_started, y = value)) + |
|  | facet\_wrap(~variable, scales = "free\_y") + |
|  | geom\_line(colour = "lightblue", alpha = 0.5) + |
|  | stat\_rollapplyr(width = 24, colour = "white") + |
|  | scale\_y\_continuous(label = comma) + |
|  | labs(x = "") + |
|  | ggtitle("Summary of Twitter information sampled since May 2018", |
|  | "White line is 24 hour moving average; blue line is original data") |
|  | return(p) |
|  | }) |
|  |  |
|  | #----------------render graphics--------------------- |
|  | output$wcp <- renderImage({ |
|  | # This could be done with just renderPlot() but that doesn't work for fonts. |
|  | # See https://stackoverflow.com/questions/31859911/r-shiny-server-not-rendering-correct-ggplot-font-family |
|  | # So unfortunately we need all this palava |
|  |  |
|  | # Read myImage's width and height. These are reactive values, so this |
|  | # expression will re-run whenever they change. |
|  | width <- session$clientData$output\_wcp\_width |
|  | height <- session$clientData$output\_wcp\_height |
|  |  |
|  | # For high-res displays, this will be greater than 1 |
|  | pixelratio <- session$clientData$pixelratio |
|  |  |
|  | # A temp file to save the output. |
|  | outfile <- tempfile(fileext='.png') |
|  |  |
|  | # Generate the image file |
|  | png(outfile, width = width \* pixelratio, height = height \* pixelratio, |
|  | res = res \* pixelratio) |
|  | par(mai=c(0,0,0,0), bg = "grey50", family = "FreeSans") |
|  | n <- nrow(hash\_data()) |
|  | wordcloud(hash\_data()$hashtag, |
|  | hash\_data()$freq, |
|  | random.order = FALSE, |
|  | ordered.colors = TRUE, |
|  | colors = inferno(n, direction = -1)) |
|  | dev.off() |
|  |  |
|  | # Return a list containing the filename |
|  | list(src = outfile, |
|  | width = width, |
|  | height = height |
|  | ) |
|  | }, deleteFile = TRUE) |
|  |  |
|  | output$tweeters <- renderImage({ |
|  | width <- session$clientData$output\_tweeters\_width |
|  | height <- session$clientData$output\_tweeters\_height |
|  |  |
|  | pixelratio <- session$clientData$pixelratio |
|  |  |
|  | outfile <- tempfile(fileext='.png') |
|  |  |
|  | png(outfile, width = width \* pixelratio, height = height \* pixelratio, |
|  | res = res \* pixelratio) |
|  | print(tweeters\_plot()) |
|  | dev.off() |
|  |  |
|  | # Return a list containing the filename |
|  | list(src = outfile, |
|  | width = width, |
|  | height = height |
|  | ) |
|  | }, deleteFile = TRUE) |
|  |  |
|  | output$batches <- renderImage({ |
|  | width <- session$clientData$output\_batches\_width |
|  | height <- session$clientData$output\_batches\_height |
|  |  |
|  | pixelratio <- session$clientData$pixelratio |
|  |  |
|  | outfile <- tempfile(fileext='.png') |
|  |  |
|  | png(outfile, width = width \* pixelratio, height = height \* pixelratio, |
|  | res = res \* pixelratio) |
|  | print(batches\_plot()) |
|  | dev.off() |
|  |  |
|  | # Return a list containing the filename |
|  | list(src = outfile, |
|  | width = width, |
|  | height = height |
|  | ) |
|  | }, deleteFile = TRUE) |
|  |  |
|  |  |
|  | #-------------------render data tables-------------------- |
|  | output$hashn <- renderText(paste0("A sample of ", |
|  | sum(as.numeric(hash\_data\_full()$freq)), |
|  | " hashtags.")) |
|  | output$hashes <- renderDataTable(hash\_data(), options = list(dom = 't')) |
|  |  |
|  | output$retweets <- renderDataTable(retweets\_data(), options = list(dom = 't')) |
|  | }) |

Shiny App – UI.R

|  |
| --- |
| library(shiny) |
|  | load("top\_ten\_lang.rda") |
|  |  |
|  |  |
|  | shinyUI(fluidPage( |
|  | tags$style(HTML("@import url('https://fonts.googleapis.com/css?family=Roboto'); |
|  | @import url('https://fonts.googleapis.com/css?family=Prosto One'); |
|  | ")), |
|  | tags$head( |
|  | tags$link(rel = "stylesheet", type = "text/css", href = "my\_styles.css"), |
|  | tags$link(rel = "canonical", href="http://shiny.ellisco.com.au/twitter-monitor/") |
|  | ), |
|  |  |
|  | # Application title |
|  | titlePanel("A representative random sample of tweets"), |
|  |  |
|  |  |
|  | sidebarLayout( |
|  | sidebarPanel( |
|  | dateRangeInput('date', |
|  | label = 'Choose a date range:', |
|  | start = Sys.Date() - 2, |
|  | end = Sys.Date() - 1, |
|  | min = "2018-05-17", |
|  | max = Sys.Date() |
|  | ), |
|  |  |
|  | conditionalPanel("input.tabs == 'Hashtags' | input.tabs == 'Popular retweets'", |
|  | radioButtons( |
|  | "langs", |
|  | "Choose a language", |
|  | choices = top\_ten\_lang, |
|  | selected = "en"), |
|  | dataTableOutput("hashes") |
|  | ) |
|  | ), |
|  |  |
|  | # Show a plot of the generated distribution |
|  | mainPanel( |
|  | tabsetPanel(id = "tabs", |
|  | tabPanel("Hashtags", |
|  | imageOutput("wcp", height = "600px"), |
|  | textOutput("hashn") |
|  | ), |
|  | tabPanel("Popular retweets", |
|  | dataTableOutput("retweets") |
|  | ), |
|  | tabPanel("Tweeters", |
|  | imageOutput("tweeters") |
|  | ), |
|  | tabPanel("Sampling", |
|  | imageOutput("batches"), |
|  | p("The image above shows the rate at which new data is being |
|  | sampled from Twitter into the database, in 30 second bursts. |
|  | For example, around 1,000 to 3,000 tweets are loaded in each |
|  | burst. Sampling bursts take place once an hour at a random |
|  | time."), |
|  | p("The actual sampling rate is unknown because it depends on the |
|  | proportion of tweets provided in Twitter's streaming API which |
|  | varies over time. |
|  | Studies have estimated this to be between 1% and 40% of all |
|  | actual tweets. The database behind this app then has a sample of |
|  | 1/120th of those; so the sample here probably ranges between one |
|  | in 12,000 and 1 in 300."), |
|  | p("The number of tweets in the sample show both daily and weekly periodicity; |
|  | peak time is around 15:30 UTC each day.") |
|  | ) |
|  | ) |
|  | ) |
|  | ) |
|  | ) |
|  | ) |

So what have I learned about Twitter (as opposed to about Linux administration) from the exercise? No time to explore in much depth right now, but some of the interesting things include:

* Tweets have a daily cycle, peaking at around 15:30 UTC each day (this assumes that the sampling ratio in the Twitter sample stream is roughly constant; which I think is likely as otherwise why would we see this seasonality).
* The most tweeted hashtags all relate to teen-oriented popular music. My filter bubble isn’t so much a liberal-v-conservative one as something relating to different interests to most people in the world altogether. The things that dominate my own Twitter feed are not even faintly representative of Twitter as a whole (I expected this with regard to statistical computing of course, but it was interesting to find out that even US politics hardly makes a dent in the most common tweets/retweets in any particular day, compared to popular retweets such as “If the Cleveland Cavaliers win the 2018 NBA finals I’ll buy everyone who retweet’s this a jersey…” (sic) – 1.1 million retweets – and several suspiciously similar variants)
* If you ignore tweets in Japanese, Korean, Thai and Arabic script you are missing three of the top seven languages on Twitter. Any serious analysis needs to find a way to bring them on board (my first obstacle in this was getting a font that could represent as many different scripts and emojis as possible without knowing in advance the language; in the end I opted for GNU FreeFont

There’s some interesting statistical challenges with using this database for inference that I might come back to. For example, I could use a small amount of auxiliary information such as the 1.1 million retweets of that Cleveland Cavaliers jersey tweet and compare it to the 105 times I found the tweet in my sample; and deduce that my sample is about 1 in 10,000 of the full population of tweets. This is consistent with the sample stream being a genuine 1% sample, of which I collect 1/120th (30 seconds every hour). So I should be able to treat my sample as a 1/12,000 sample of the whole population, clustered by the 30 second window they are in. Something for later.